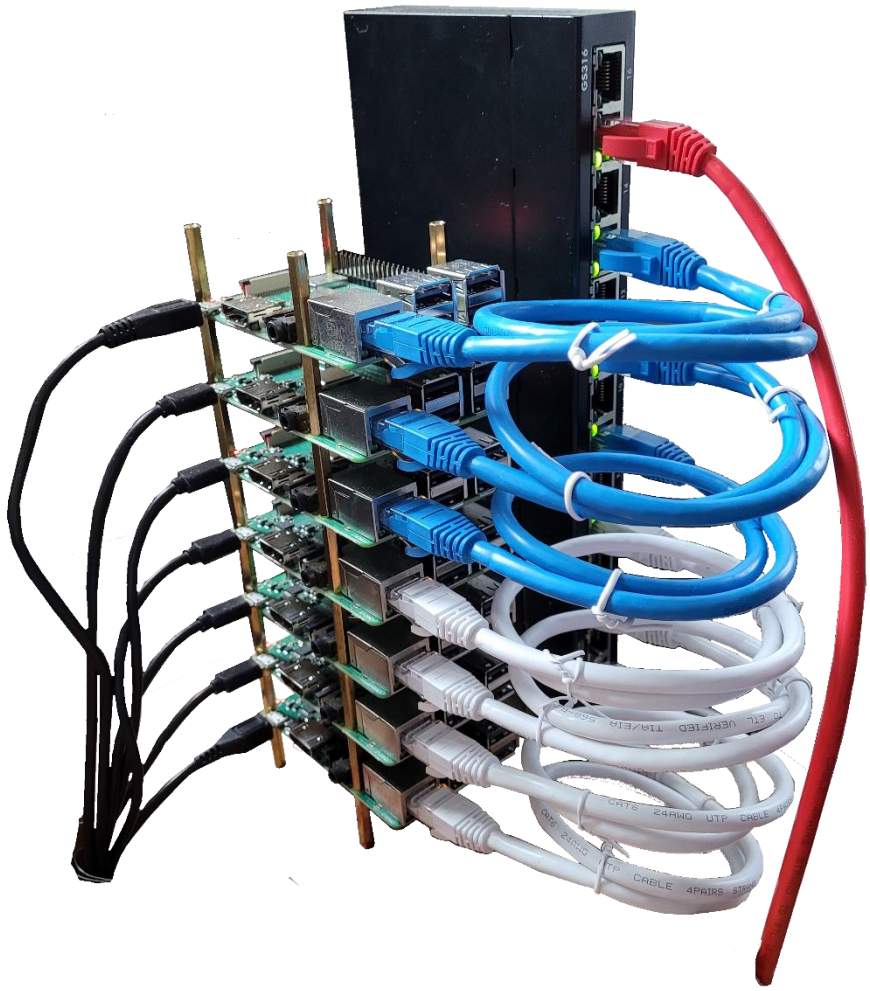
Parallel Computing Final Project

Julia Set Fractal Generation & Gnome Sorting on Pi Cluster

Wojciech Zacherek CM 601



Introduction

Hardware

I used three computing platforms to evaluate my solutions: my personal computer, my Pi Cluster, and the Beowulf Computing Cluster.

My personal computer uses an Intel Core i7-6700K CPU clocked at 4.00 GHz. This system has 16 GB of RAM available at 2133 MHz.

My Pi Cluster uses 7 Raspberry Pi 3B+ (referred to as Pi henceforth), each using 64GB microSD card to store the OS and programs. For this project, the cluster was flashed with Raspberry Pi OS Lite – a minimal installation of the raspberry pi OS with basic networking functionality. To connect the units, I used 3ft Cat5 ethernet cables and a 16 port Netgear unmanaged switchbox. Pi routing was handled by a Linksys WRTU54G-TM router (the old one), assigning a static IP to each Pi.

The Beowulf Computing Cluster offers 180 processors subdivided into six groups with 30 processors each.

Software

I used VSCode for programming. For testing code before deployment, I used Window's Subsystem for Linux (WSL) on my personal computer. WSL interfaced nicely with MPI and offered native libraries.

Notes on Setting up the Pi Cluster

Setting up the cluster can be a hassle and getting MPI up and running can be difficult. To help mitigate confusion and save some time, this section will give an outline on how I setup my cluster. These instruction may change in the future, so I would treat this section as pseudocode: good outline of procedure, but implementation will vary.

First, we need to choose the proper OS. To minimize hardware issues, I would recommend using one of the raspberry pi distributions. These distributions can be downloaded manually or automatically using the Raspberry Pi Imager[1]. I would recommend the "Raspberry Pi OS Lite" distribution, since it is very light weight and has all the necessary component to get started.

Next is networking the Pi's together. I found two ways to network Pi's, but only implemented one due to time constraints, however both should work:

1. The first method involves an external router. Most routers have automated IP assignments, but I would recommend using static IP assignment. This will make using MPI later much easier since you will always know the local IP of each Pi.
2. The second method treats one of the Pi's as a router. There are ubuntu packages that allow the Pi to take on the role of a router without the need of one. In this case all the Pi's can be connected over a network switch.

As an additional resource, I've listed mt Git Repo: https://github.com/wojtek-zacherek/PiCluster .

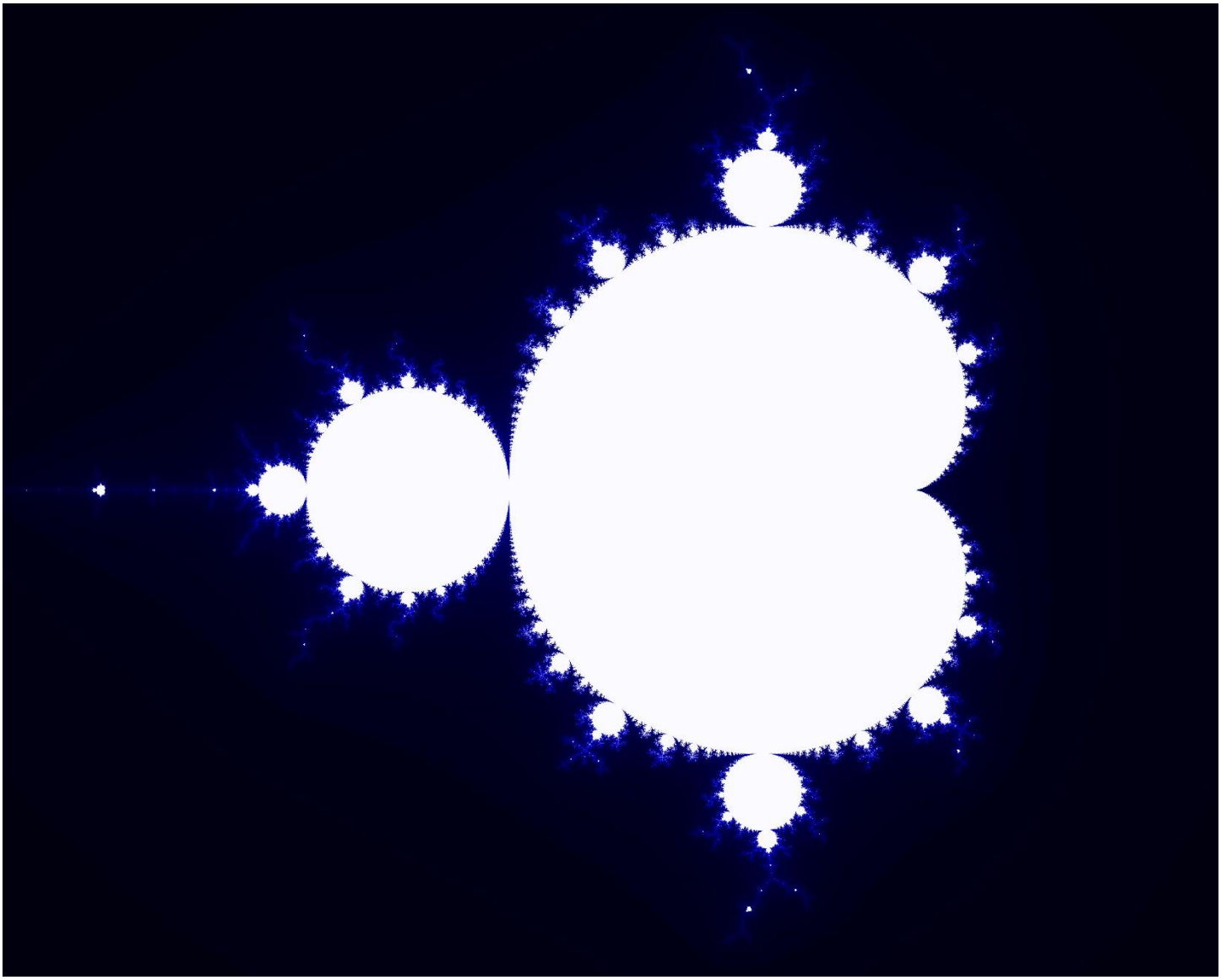
[1] Raspberry Pi OS – Raspberry Pi

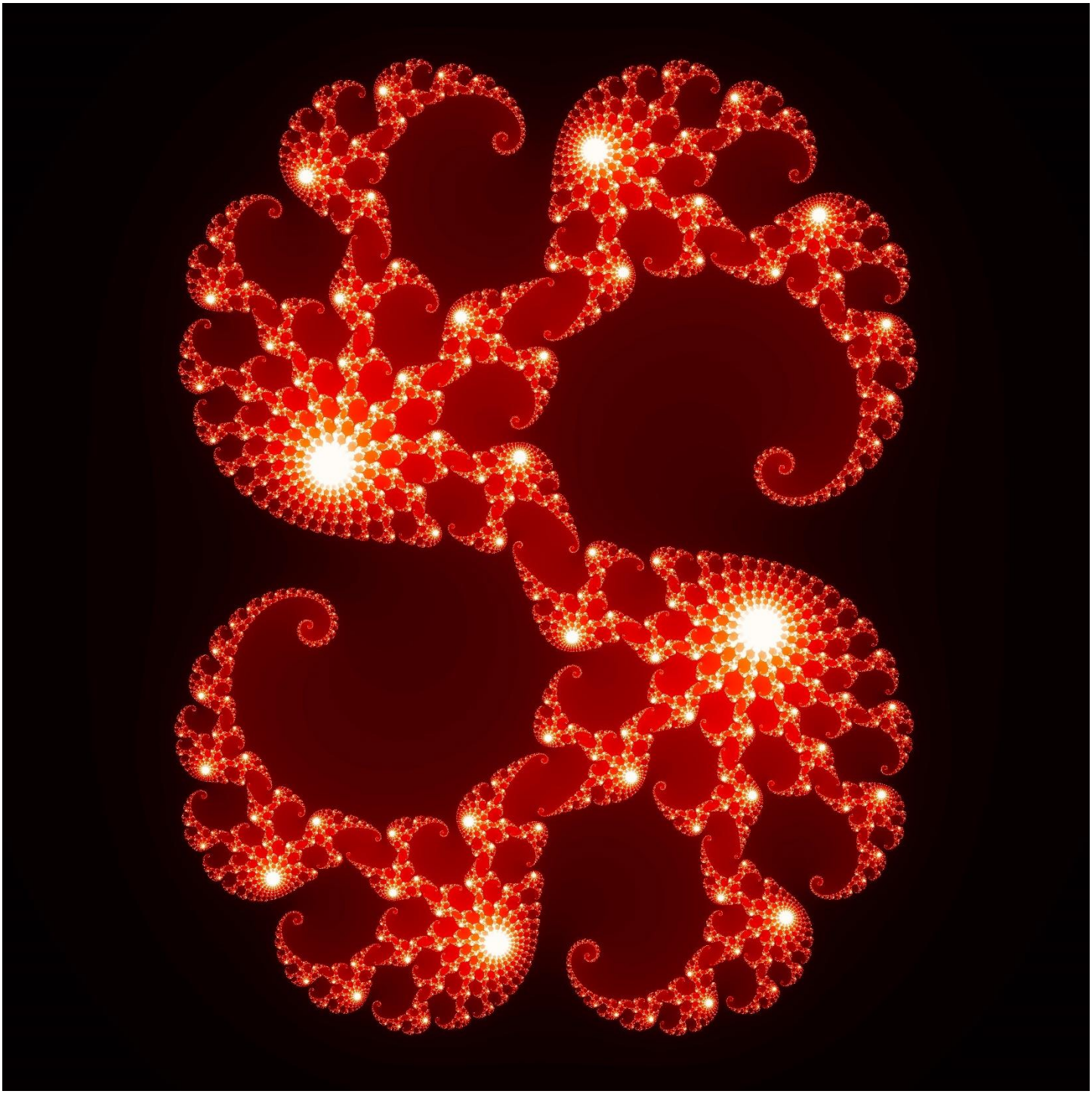
Julia Set Fractal Generation

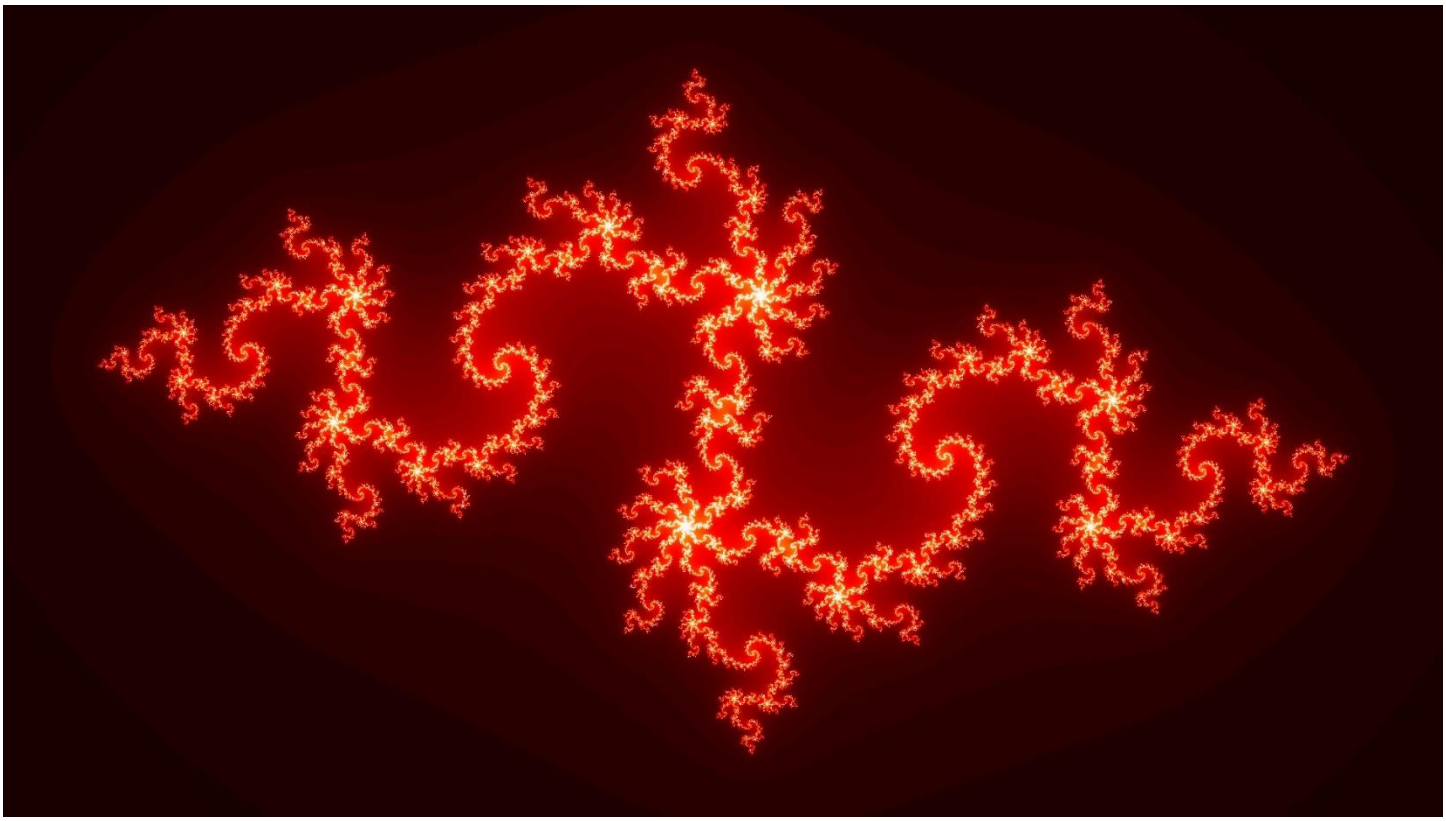
The Julia set fractal is plot of points that follow a set of equations. The more popular example is the Mandelbrot fractal. Typical depictions of these fractals utilize a color gradient to better represent when points stop following the equations.

I implemented MPI parallelism by dividing up the plot among the processors. My current parsing distributes point column wise.

Increasing the number of processors had mixed results. The optimal best-case is runtime decreasing by a factor proportional to the number of processors. However, practice showed improvement lower than that. This stems from the nature of the fractal problem. For any given set of equations, regions that drop-off quickly will be computed quickly, while longer-lasting regions will take up more computing time. I currently don't have an intelligent way to handle situation like this other than manually modifying the distributions after viewing the result.







Gnome Sort

Gnome sort is a simple sorting algorithm that mimics the sorting patterns of the gnomes (allegedly fictitious creatures). This sort is similar to the insertion sort, usually has a longer runtime than the insertion sort, but has the same worst-case and best-case performance characteristics.

This algorithm works according to the following description taken from Dick Grune:

> Here is how a garden gnome sorts a line of flower pots.
> Basically, he looks at the flower pot next to him and the previous one; if they are in the right order he steps one pot forward, otherwise, he swaps them and steps one pot backward.
> Boundary conditions: if there is no previous pot, he steps forwards; if there is no pot next to him, he is done.[2]

A simple pseudocode example looks like the following:

```
procedure gnomeSort(a[]):
    pos := 0
    while pos < length(a):
        if (pos == 0 or a[pos] >= a[pos-1]):
            pos := pos + 1
        else:
            swap a[pos] and a[pos-1]
            pos := pos - 1
```
[3]

I used MPI similarly to previous projects:

- Each processor receives a portion of the unsorted data.
- Local sorting occurs on each set of data.
- The processors return data in a tree-like fashion, up until the parent processor has the complete data.

Results Nomenclature

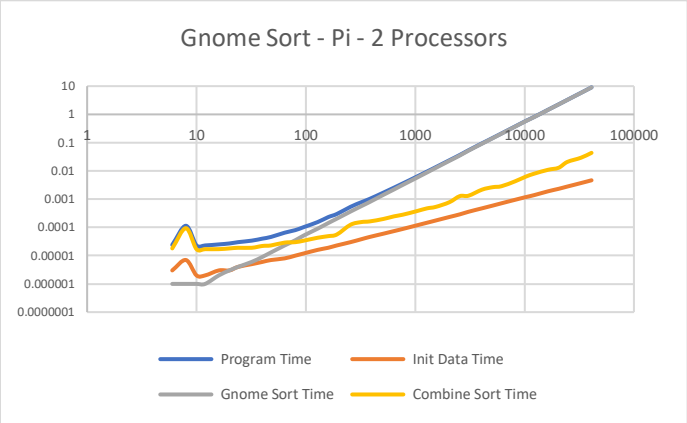
For the timing charts that varies the number of N elements:

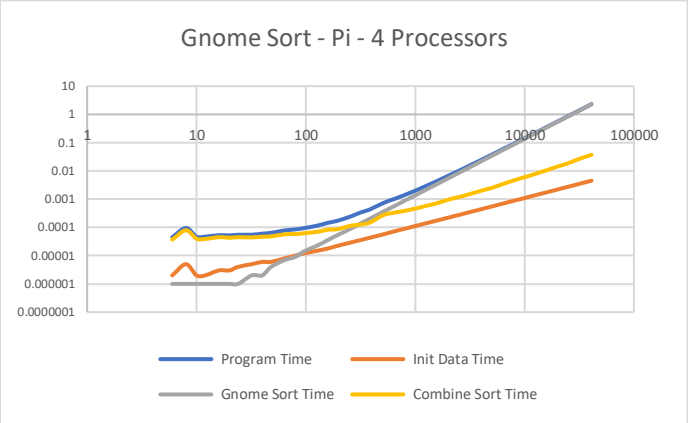
- Program Time: the total runtime of a single sort, from the creation of the array to it's release.
- Init Data Time: how much time is required to setup the array.
- Gnome Sort Time: how much time the Gnome sort takes on processor 0.
- Combine Sort Time: how much time it takes to combine the results of each processor.

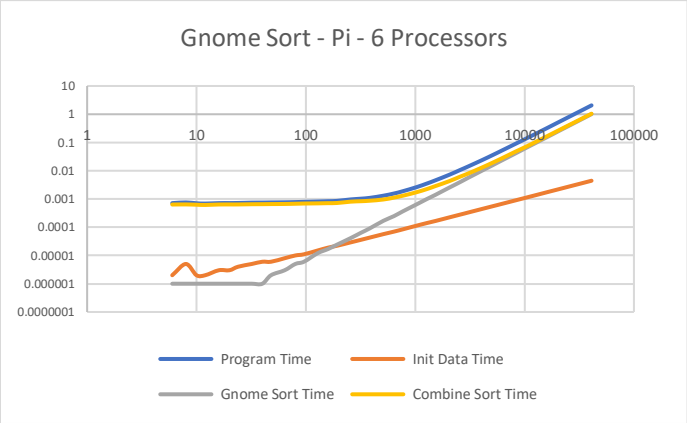
[2] "Gnome Sort - The Simplest Sort Algorithm". *Dickgrune.com*
[3] https://en.wikipedia.org/wiki/Gnome_sort

Pi Results

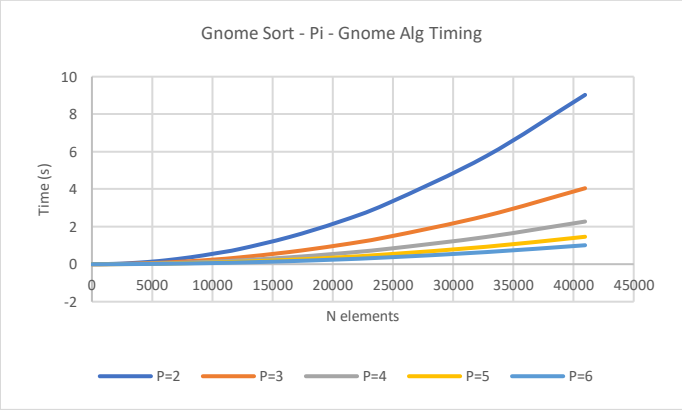
For a low number of processors, the Gnome algorithm defines the program time. However, when we reach 6 processors, the combination-sort time takes longer than the Gnome sort itself.

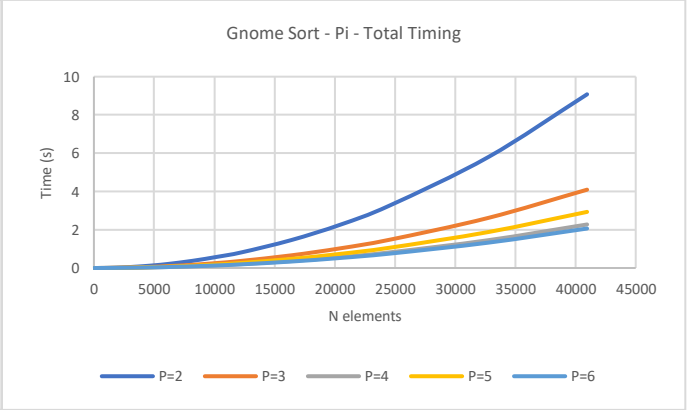






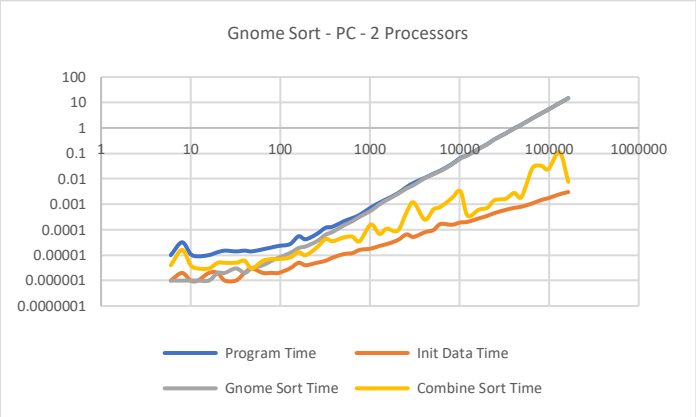
The increase in performance follows the typical idea of “more processors leads to more computing, but at a lower efficiency.”

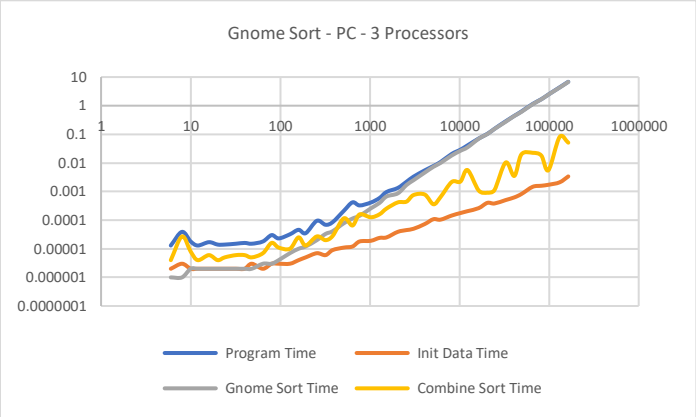


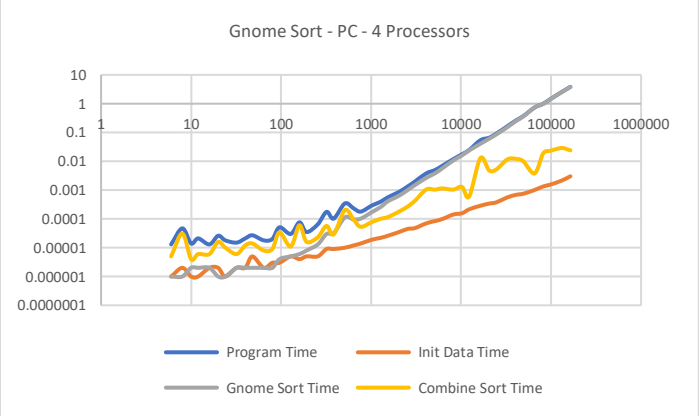


PC - Single Multicore Processor Results

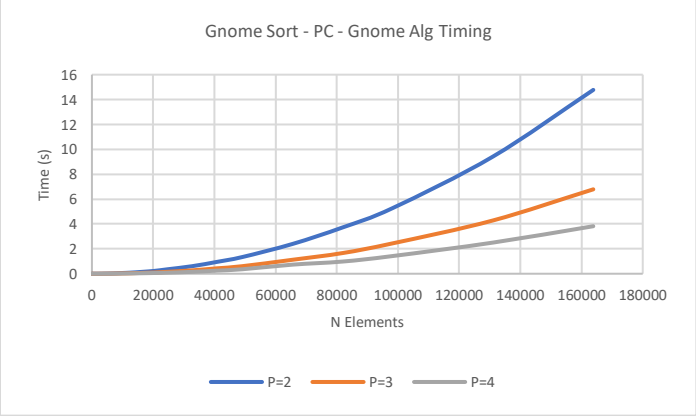
My PC showed very dynamic timing results during the combination-sort of the program. This may have been caused by other programs running in the background and utilizing some of my computer's resources.

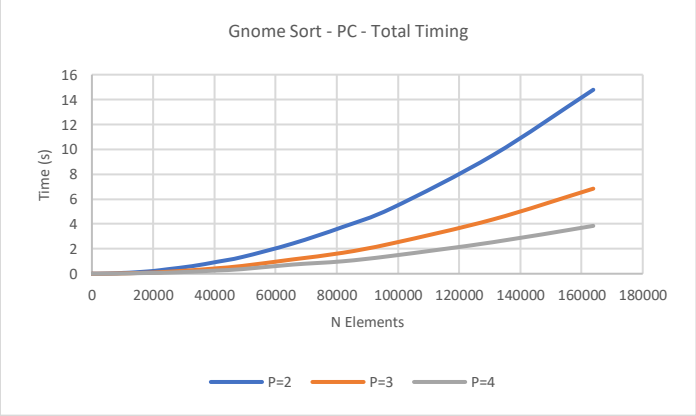






The increase in performance follows the typical idea of "more processors leads to more computing, but at a lower efficiency."

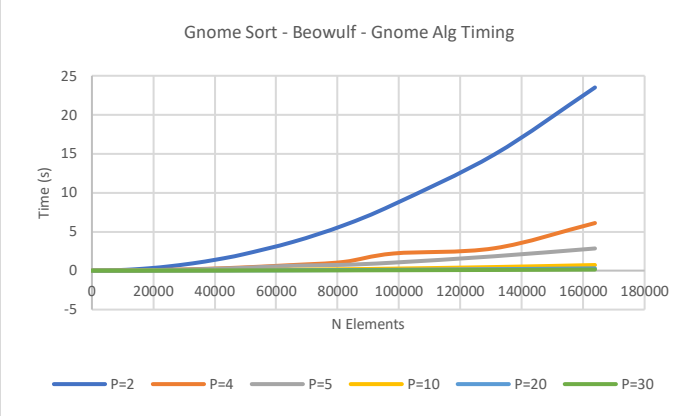


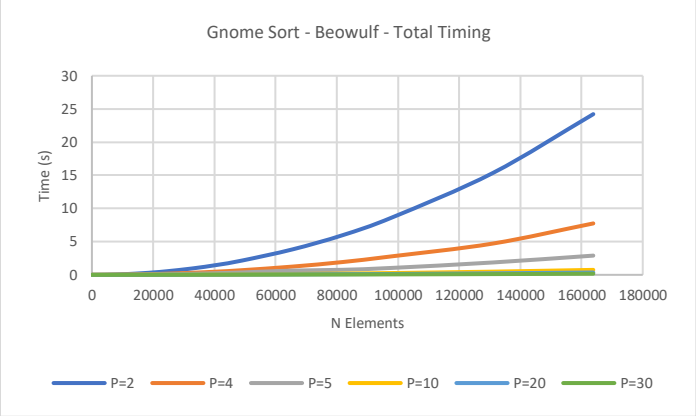


Beowulf Cluster Results

For the majority of runs, the Gnome sorting section of the program dominates computation time. However, as the number of processors increase, the combination-sorting time begins to dominate for the shorter lists.

The increase in performance follows the typical idea of “more processors leads to more computing, but at a lower efficiency.”


Gnome Sort - Beowulf - Gnome Alg Timing
Time (s)
25
20
15
10
5
0
-5
0
20000
40000
60000
80000
100000
120000
140000
160000
180000
N Elements
P=2
P=4
P=5
P=10
P=20
P=30


Gnome Sort - Beowulf - Total Timing
Time (s)
30
25
20
15
10
5
0
0
20000
40000
60000
80000
100000
120000
140000
160000
180000
N Elements
P=2
P=4
P=5
P=10
P=20
P=30

Overall Analysis

The Gnome sort is not a good sorting algorithm, but application results clearly vary. I was surprised to find my Pi Cluster outperforming both my personal computer and the Beowulf Cluster (for the number for processors that my cluster can run). If we take the first situation, when P=2, we get the following timing characteristics:

- Pi: 9.02 seconds
- PC: 14.77 seconds
- Beowulf: 23.5 seconds

Even up to P=6, the Pi Cluster consistently completes quicker than the other two. My intuition suggests that the following reasons for this performance discrepancy:

- I designed my Pi Cluster extremely lightweight compared to the Beowulf; even though the Beowulf cluster has better processors than my Pi's, the overhead and other processes running on Beowulf outweigh this difference.
- A similar idea for my PC, especially since it was running windows OS and other background programs.

Conclusion

This project and timing analysis of my programs verified what I've learned in this course. Namely:

- Blindly increasing the number of processors usually doesn't lead to linear speedup.
- Inter-process communication can become a bottleneck, so smart design is required.

Appendix